tensorflow改写工作思路

笔记本： Lab at UCLA
创建时间： 10/18/2017 12:56 PM **更新时间：** 10/20/2017 9:05 AM
作者： smshhw@pku.edu.cn
URL： about:blank

1 forward部分，需要在Layer中增加一个conv_mask层：
需要：1 看懂base，知道layer的构成
2 看懂_Conv，知道conv layer的构成
3 因为mnist是2维data，所以我们在Conv2D里面添加一个函数，叫conv_mask，来实现forward pass时的功能
为定义conv_mask，需要借鉴之前的**matlab代码**：主要工作有：
getMu
getMask
做点乘，取max值，形成输出（这一步是最简单的）

疑问：
conv_mask
放在conv2D类吗?
trainble=True,

在layer类中，有losses 和 updates的properties，是不是可以利用一下?

其实conv_mask层有两种看法：（我先尝试了1）
1 把它和之后的FC层合起来看，那就是先一个mask再一个dense，这样的话属于一个Dense类。但它loss的定义需要改啊。是不是需要一个新的Class?
Dense类中形成output时，layer.apply()
apply会直接用self.__call__,这个函数实际上真正运行的是self.call
终于溯源成功了！"the logic of the layer lives here"

```
    def call(self, inputs, **kwargs):
      """The logic of the layer lives here.

      Arguments:
        inputs: input tensor(s).
       **kwargs: additional keyword arguments.

      Returns:
        Output tensor(s).
      """
      raise NotImplementedError
```

这个call函数在每个子layer类里都会被override，具体到Dense类里
（即core.py的130行开始）

Note: if the `inputs` tensor of the layer.dense has a rank greater than 2, then it is flattened prior to the initial matrix multiply by `kernel`.

```
`outputs = activation(inputs.kernel + bias)`
Where `activation` is the activation function passed as the `activation`
argument (if not `None`), `kernel` is a weights matrix created by the layer,
and `bias` is a bias vector created by the layer
(only if `use_bias` is `True`).

Note: if the input to the layer has a rank greater than 2, then it is
flattened prior to the initial matrix multiply by `kernel`.
```

现在的麻烦是：我不知道是不是可以直接在dense里面的len(output_shape)>2里面改
————**我觉得可以！**（相信官网的描述！）
那么找准位置以后，剩下的就是写出getMask了
写getMask中的疑点：MATLAB代码中的postTemp是干什么的?

可以把getMu函数放在getMask里
getMu的改写：

我保留了1：h为np.arange(1, h)，但是python和matlab之间的index方式不一样，我担心之后可能会有地方出问题。。。
还有一点是，cnn_mnist中，所有的shape参数的顺序都把batch size放在第一个，所以我改写的时候也把matlab中的顺序改过来了。但也有可能有遗漏的。
np.multiply和np.dot的选择，应该所有的 .* 都是可以用np.mutilply带代替的（element wise)
np.tile实现的是matlab中多维情况下repmat

一个小**不确定**：在getMu else中的sqrtvar计算中的[h*w, 1, 1]最后的1应该还是batchSize把？我把它在numpy中移到第一个了
一个**问题**：`sqrtvar = np.tile(sqrtvar, [depth, batchS])`
这里 [depth, batchS] 是原来的顺序，我也先改了一下。不知道需不需要？？
getMu中：`mu_x=reshape(tmp(mu_x),[1,1,depth,batchS])`
这里的tmp(mu_x)是什么意思？——ANS: index！改成tmp[mu_x]即可！
问题：mask构造中的repmat函数的参数[h, w, 1]中的1是深度还是batchS？这里我没有改代码
问题：layer.weights[3]是什么参数？？

最大的问题：MatConvNet中，**layer有posTemp，filter, weights这些参数**
目前最大的障碍。其他基本都搞定了。
可以从layer作为inputs有什么入手：看上一层max_pooling2d的ouput有些什么
看_Pooling2D class 中的call

(它的loss怎么解决晚点再说，我先把forward pass给写了！)

2 把它单独拎出来，那它就更像relu,pooling这样的一个操作，还没有自己的参数，mu和mask都是从上一个layer中计算出来的，output也就是Input的经过类似relu操作后的一个layer，成为dense层的Input，感觉这样理解会更好操作一点。但不知道计算loss和gradient的时候会怎么样。

---

2 backward的部分，需要的工作复杂一些，还有点不清楚怎么搞，因为训练的时候，tensorflow的方法是：

```
if mode == tf.estimator.ModeKeys.TRAIN:
  optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
  train_op=optimizer.minimize(
    loss=loss,
    global_step=tf.train.get_global_step())
  return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)
```

我们有一个总的loss

要理解它们的输入、输出 和 中间过程

optimizer, 是optimzer函数的bass class，这里面告诉我们，对于我们的这个方法，不能直接用minimize()了，可以先**compute_gradients()**，再process，在apply_gradients()。需要改的文件至少有**compute_gradients()**和apply，它们在mask上下是有变化的

初步实现时，忽略gate gradients, slots

```
### Processing gradients before applying them.

Calling `minimize()` takes care of both computing the gradients and
applying them to the variables. If you want to process the gradients
before applying them you can instead use the optimizer in three steps:

1. Compute the gradients with `compute_gradients()`.
2. Process the gradients as you wish.
3. Apply the processed gradients with `apply_gradients()`.

Example:

```python
# Create an optimizer.
opt = GradientDescentOptimizer(learning_rate=0.1)
```
```

```
# Compute the gradients for a list of variables.
grads_and_vars = opt.compute_gradients(loss, <list of variables>)

# grads and vars is a list of tuples (gradient, variable). Do whatever you
# need to the 'gradient' part, for example cap them, etc.
capped_grads_and_vars = [(MyCapper(gv[0]), gv[1]) for gv in grads_and_vars]

# Ask the optimizer to apply the capped gradients.
opt.apply_gradients(capped_grads_and_vars)
```

关于Optimizer类中的minimize()函数：

```
"""Add operations to minimize `loss` by updating `var_list`.

This method simply combines calls `compute_gradients()` and
`apply_gradients()`. If you want to process the gradient before applying
them call `compute_gradients()` and `apply_gradients()` explicitly instead
of using this function.
```

cnn_mnist中的minimize里有一个参数global_step，这里解释一下：
global_step: Optional `Variable` to increment by one after the variables have been updated.

我们还是可以用minimize()函数，需要改的只是compute_gradients这个函数，做个if conditional？而真正的求gradient的操作在函数gradients_impl里面。这是一个数学的operation。

想一想是在底层（gradients_impl）里改，还是在compute_gradients里面改————ANS:应该还是要在compute_gradients里面改，设置两种情况，正常的时候就走gradients.gradients，进行数学运算；在conv层（即从FC回到Relu的时候），gradients的计算会有所不同。

如果这边有困难，可以参考CS231n作业中关于Backpropagation的部分，看看底层操作应该怎么搞

Backpropagation 思路
需要修改的就是

```
train_op = optimizer.minimize(
        loss = loss,
        global_step=tf.train.get_global_step())
```

这一步，而这一步的关键在于optimizer.minimize函数
这个函数跟gradient descent什么的没有关系，它是在Optimizer类中就定义了的
minimize分为**compute gradients** 和 apply gradients两部分，
需要修改的就是这两个部分

另一个方法是：直接把minimize用compute+process + apply来代替

明天跟师兄确认一下：往回算的时候，mask情况的求导是就一次吗？之后就一直是普通的求gradients了吧？
如果是这样就好办了，只要在compute gradients里面分个类就行了。
这里的mask用的是之前的mask吗？可以用forward方法重新计算一遍（反正只用一次）

gen_nn_ops里有max pooling的gradient计算，也许有参考意义

dense 改成our_dense
改过的文件存在一个单独的文件夹里

测试写的对不对：正负样本的mask，
能不能打出mask之前和之后的中间结果，还有原来的feature map——（spider?）

先跑一下原来的程序，设置断点，看看是不是调用了**conv2d**